Journal of Uncertain Systems
Vol.10, No.1, pp.72-80, 2016
Online at: www.jus.org.uk

When Can We Simplify Data Processing: An Algorithmic Answer

Julio Urenda[1], Olga Kosheleva[1], Vladik Kreinovich[1],[*] Berlin Wu[2]
[1] *University of Texas at El Paso, El Paso, TX 79968, USA*
[2] *Department of Mathematical Sciences, National Chengchi University, Taipei 116, Taiwan*

Received 11 June 2015; Revised 5 July 2015

Abstract

In many real-life situations, we are interested in the values of physical quantities $x_1, \ldots, x_n$ which are difficult (or even impossible) to measure directly. To estimate these values, we measure easier-to-measure quantities $y_1, \ldots, y_m$ which are related to the desired quantities by a known relation, and use these measurement results to estimate x_i. The corresponding data processing algorithms are sometimes very complex and time-consuming, so a natural question is: are simpler (and, thus, faster) algorithms possible for solving this data processing problem? In this paper, we show that by using the known Tarski-Seidenberg algorithm, we can check whether such a simplification exists and, if it exists, produce this simplification.

Keywords: data processing, indirect measurements, measurement uncertainty, algorithm simplification

1 Formulation of the Problem

Need for data processing. In many practical situations, we are interested in a quantity which is difficult or even impossible to measure directly. For example:

- while it is possible to directly measure a distance to the Moon – by sending a laser signal to a Moon-based reflector and measuring the time that it takes to get back – there is no easy way to directly measure a distance to the faraway star;
- while we can directly measure the amount of oil in a reservoir, it is not possible to directly measure the overall amount of oil in an oil field.

In all such situations, when we cannot directly measure the values of the quantities-of-interest $x_1, \ldots, x_n$, we:

- find auxiliary easier-to-measure quantities $y_1, \ldots, y_m$ which have known relation with x_i, i.e., for which $y_i = f_i(x_1, \cdots, x_n)$ for some known function f_i;
- measure the values y_i, and then
- find the values of the desired quantities by solving the corresponding system of equations

$$f_1(x_1, \cdots, x_n) = y_1;$$

$$f_2(x_1, \cdots, x_n) = y_2;$$

$$\ldots$$

$$f_m(x_1, \cdots, x_n) = y_m.$$

[*]Corresponding author.
Emails: jcurenda@utep.edu (J. Urenda), olgak@utep.edu (O. Kosheleva), vladik@utep.edu (V. Kreinovich), berlin@nccu.edu.tw (B. Wu).

Computations needed to reconstruct the desired values x_i from the measurement results y_j form what is known as *data processing.*

Examples. To find a distance to a faraway star, we can measure the angle to this star from different Earth locations – or, better yet, from different points on the Earth's orbit. Then, we can use trigonometric formulas to estimate the actual orbit. Even more complex computations are needed to estimate the amount of oil in a given oil field.

Data processing algorithms are often time-consuming. In many practical situations, the dependencies f_i between physical quantities are non-linear. In such situations, to find the desired quantities x_i, we need to solve systems of non-linear equations.

It is known that the general problem of solving a system of non-linear equations – even quadratic equations – is NP-hard; see, e.g., [4]. This means, crudely speaking, that unless P=NP (which most computer scientists believe to be impossible), no algorithm is possible that would always solve all such systems in feasible time. In other words, algorithms for solving such systems often require a large amount of computation time.

This is especially critical in real-time situations, when we use the estimates of the dynamically changing quantities x_i to make decision – e.g., in control situations and in medical decision making.

The problem becomes even more complex if we take measurement uncertainty into account. The above equations describe the ideal case, when the measurement uncertainty can be ignored and the measured values y_i are exactly equal to the values $f_i(x_1, \cdots, x_n)$. In practice, there is always some measurement uncertainty.

Usually, we know an upper bound Δ_i on the inaccuracy of each measurement. This means that the measurement result y_i can differ from the actual (unknown) value $f_i(x_1, \cdots, x_n)$ by no more than Δ_i. In this case, instead of the above system of equations, we have a system of inequalities:

$$y_1 - \Delta_1 \le f_1(x_1, \cdots, x_n) \le y_1 + \Delta_1;$$

$$y_2 - \Delta_1 \le f_2(x_1, \cdots, x_n) \le y_2 + \Delta_2;$$

$$\cdots$$

$$y_m - \Delta_m \le f_m(x_1, \cdots, x_n) \le y_m + \Delta_m.$$

This measurement uncertainty makes the corresponding computational problem even more complex: namely, as shown in [4], this problem becomes NP-hard even in the simplest case, when all the dependencies $f_i(x_1, \cdots, x_n)$ are linear.

While the general problem is NP-hard, specific problems can be simplified. While the general problem is hard to solve, there are many specific measurement situations in which we can feasibly estimate the desired quantities x_i based on the measurement results.

In mathematical terms, a specific situation means that instead of generic functions $f_i(x_1, \cdots, x_n)$, we have a given family of functions $F_i(x_1, \cdots, x_n, c_1, \cdots, c_k)$ with parameters $c_1, \ldots, c_k$.

Resulting problem. For each specific class of measurement situations, it is desirable to find out beforehand whether for this class of situations, a feasible data processing algorithm is possible, and, if such an algorithm is possible, design such an algorithm.

It is important to notice that:

- while data processing itself is often on-line, so that the resulting data processing algorithm itself should be fast,
- the design of the corresponding algorithm can be off-line, so it is tolerable if the method for designing this algorithm requires a long computation time.

What we do in this paper. In this paper, we show how to use the known Tarski-Seidenberg algorithm to solve the above problem.

Note about the fuzzy case. In some cases, in addition to the upper bounds Δ_i on the measurement errors, experts can also tell which values within these bounds are more probable and which are less probable. Experts usually describe this additional information by using imprecise ("fuzzy") words from natural language; for example, an expert can say "the measurement error is most probably much smaller than Δ". To represent this knowledge in precise terms, it is reasonable to use techniques which were specifically designed for representing this knowledge – namely, *fuzzy techniques*; see, e.g., [3, 7, 10]. In this approach, the expert knowledge is represented by a *membership function* $\mu_i(\Delta y_i)$ that assigns, to each value of the difference $\Delta y_i \stackrel{\text{def}}{=} y_i - f_i(x_1, \cdots, x_n)$, a degree to which, according to the expert(s), this value of measurement uncertainty is possible.

For processing fuzzy data, it is often convenient to use not the original membership functions $\mu_i(\Delta y_i)$, but their *α-cuts* ${}^\alpha\Delta_i \stackrel{\text{def}}{=} \{z : \mu_i(z) \geq \alpha\}$ corresponding to different values $\alpha \in [0, 1]$. It is known that for many data processing algorithms, the α-cut describing the result of data processing is equal to the result of applying the data processing algorithms to the α-cuts of the inputs [3, 7]. So, from the computational viewpoint, it is sufficient, for each α, to consider the case when we have interval bounds on Δy_i, namely, the bounds $\Delta_i(\alpha)$ forming the corresponding α-cut ${}^\alpha\Delta_i = [-\Delta_i(\alpha), \Delta_i(\alpha)]$.

Because of the possibility of this reduction, in the present paper, we only consider the above case of interval uncertainty.

2 Formulating the Problem in Precise Terms

What we know. We know the measurement results y_j and we know the relation between these results and the desired values $x_1, \ldots, x_n$. These relations are described in terms of equalities or inequalities. Each of these equalities and inequalities can be transformed into a form $f(x_1, \cdots, x_n, y_1, \cdots, y_m) = 0$ or $f(x_1, \cdots, x_n, y_1, \cdots, y_m) \geq 0$, for some computable function $f(x_1, \cdots, x_n, y_1, \cdots, y_m)$: indeed, each relation of the type $u = v$ or $u \geq v$ can be equivalently reformulated as $u - v = 0$ and, correspondingly, $u - v \geq 0$.

From the mathematical viewpoint, this function can be complex, e.g., it can include computing $\sin(x)$, $\exp(x)$, etc. In the computer, however, the only hardware supported elementary arithmetic operations are arithmetic operations $+$, $-$, $\cdot$, and $/$. Thus, any actually computed function is a composition of such operations – i.e., a rational function (a ratio of two polynomials). For example, the values of the functions $\sin(x)$ and $\exp(x)$ are computed, in most computers, by computing the sum of the first few terms in their Taylor expansions – i.e., as polynomials.

We can also have branching, when different expressions are used depending on the values x_i and y_j; this branching is also done by comparing the values of two comparable expressions. Let us describe these computations in precise terms.

It should be noted that the functions $f(x_1, \cdots, x_n, y_1, \cdots, y_m)$, which are used in describing the relation between the measurement results y_j and the desired values x_i, rarely use loops. Loops are often used to *solve* these systems, i.e., to find the desired values x_i from the measurement results y_j, but not to describe the relation. Let us therefore explicitly describe the corresponding notion of a loop-less algorithm.

Definition 1. *Let N and T be natural numbers. The number N will be called a* number of inputs*, and the number T will be called the* number of computation steps*. By an* instruction I_i corresponding to step i*, we mean an expression of one of the following five types:*

- *"$r_{T+i} \to r_j \oplus r_k$", where $\oplus$ is one of the four arithmetic operations ($+$, $-$, $\cdot$, and $/$), and $j, k < T + i$;*
- *"go to Step a", where $a > i$;*
- *"if $r_j = r_k$ then go to Step a else go to Step b", where $j, k < T + i$ and $a, b > i$;*
- *"if $r_j \geq r_k$ then go to Step a else go to Step b", where $j, k < T + i$ and $a, b > i$;*
- *"return r_j", where $j < T + i$; and*
- *"stop".*

By a straightforward (loop-less) algorithm *A, we mean a sequence of instructions $I_1, \ldots, I_N$ in which the only allowed variables r_i are the ones for which the corresponding instruction I_i is of the arithmetic type "$r_{T+i} \to r_j \oplus r_k$".*

Definition 2. *Let A be a straightforward algorithm with N inputs, and let $r_1, \ldots, r_N$ be real numbers. By the* result *of applying the algorithm A to these numbers, we mean the value(s) that are obtained after a step-by-step implementation of the corresponding instructions.*

Definition 3. *We say that a function $f(v_1, \cdots, v_M)$ is straightforwardly computable if there exist constants $c_{M+1}, \ldots, c_N$ and a straightforward algorithm A for which, for every tuple $v_1, \ldots, v_M$, the result of applying A to the values $v_1, \ldots, v_M, c_{M+1}, \ldots, c_N$ coincides with $f(v_1, \cdots, v_M)$.*

Example. Let us show how a simple computation of a piece-wise linear function can be described in these terms. We want to compute a function $f(x)$ which is equal:

- to $a \cdot x + b$ when $x \le x_0$ and
- to $c \cdot x + d$ when $x > x_0$.

In this example, we have $N = 6$ inputs: $r_1 = x$, $r_2 = a$, $r_3 = b$, $r_4 = x_0$, $r_5 = c$, and $r_6 = d$. The corresponding instructions have the following form:

- the first instruction I_1 is

  "if $r_4 \ge r_1$, then go to Step 2 else go to Step 6";

  this instruction checks whether $x \le x_0$;
- the instruction I_2 is "$r_8 \leftarrow r_2 \cdot r_1$"; it computes $a \cdot x$;
- the instruction I_3 is "$r_9 \leftarrow r_8 + r_3$"; it adds b to $a \cdot x$ and thus, computes $a \cdot x + b$;
- the instruction I_4 is "return r_9";
- the instruction I_5 is "stop";
- the instruction I_6 is "$r_{12} \leftarrow r_5 \cdot r_1$"; this instruction computes $c \cdot x$;
- the instruction I_7 is "$r_{13} \leftarrow r_{12} + r_6$"; it adds d to $c \cdot x$ and thus, computes $c \cdot x + d$;
- the instruction I_8 is "return r_{13}";
- the final instruction I_9 is "stop".

In this case, we have one input $M = 1$, and five constants $c_2 = a$, $c_3 = b$, $c_4 = x_0$, $c_5 = c$, and $c_6 = d$. The above algorithm shows that the original piece-wise linear function is straightforwardly computable.

Formulation of the problem. We have relations expressed by straightforwardly computable functions; we want to check whether a simple (straightforward) algorithm is possible for estimating the desired quantities x_i based on y_j.

For example, we may want to check if we can find a linear mapping transforming y_j into x_i, or a quadratic mapping, or a piece-wise linear mapping, with some threshold separating two linear expressions. In all these examples, we have a general straightforward algorithm with unknown coefficients, and we would like to:

- check whether it is possible to find the values of the coefficients for which the algorithm always reconstructs x_i from y_j, and
- if it is possible, to actually find the values of these coefficients.

For example, in the case of a linear transformation, the parameters a_i and a_{ij} describing a general linear transformation $x_i = a_i + \sum\limits_j a_{ij} \cdot y_j$ are the desired coefficients.

There can be two versions of this problem:

- we can simply want to find *some* values x_i that satisfy all the constraints, or
- we may want to find, for each i, the exact lower bound $\underline{x}_i$ and the exact upper bounds $\overline{x}_i$ of the values x_i corresponding to possible solutions; see, e.g., [2, 6].

Let us describe all this in precise terms.

Definition 4.

- *By a* data processing problem P*, we mean a finite set of relations of the type* $f_i(x_1, \cdots, x_n, y_1, \cdots, y_m) = 0$ *or* $f_i(x_1, \cdots, x_n, y_1, \cdots, y_m) \geq 0$.
- *We say that a tuple* $y = (y_1, \cdots, y_m)$ *is* possible *for a data processing problem* P *if there exists values* $x_1, \ldots, x_n$ *for which all relations from* P *are satisfied.*
- *We say that an algorithm* A solves the data processing problem P *if for every possible tuple* y*, this algorithm returns the values* $x_1, \ldots, x_n$ *for which all relations from* P *are satisfied.*
- *We say that an algorithm* A reliably solves the data processing problem P *if for every* i*, it returns the exact lower bound* $\underline{x}_i$ *and the exact upper bounds* $\overline{x}_i$ *of the values* x_i *corresponding to all possible solutions.*

Definition 5. *By a* data processing simplification problem*, we mean the following problem:*

- given *a data processing problem* P *with straightforwardly computable relations, and a straightforward algorithm* A *with* $N \geq m$ *inputs,*
- check *whether there exist values of the parameters* $c_{m+1}, \ldots, c_N$ *for which* A *solves the data processing problem;*
- *for the cases when such values exist, compute such values* $c_{m+1}, \ldots, c_N$.

Definition 6. *By an* reliable data processing simplification problem*, we mean the following problem:*

- given *a data processing problem* P *with straightforwardly computable relations, and a straightforward algorithm* A *with* $N \geq m$ *inputs,*
- check *whether there exist values of the parameters* $c_{m+1}, \ldots, c_N$ *for which* A *reliably solves the data processing problem;*
- *for the cases when such values exist, compute such values* $c_{m+1}, \ldots, c_N$.

3 Main Results

Proposition 1. *There exists an algorithm that solves the data processing simplification problem.*

Proposition 2. *There exists an algorithm that solves the reliable data processing simplification problem.*

Comment. Our proof uses the Tarski-Seidenberg algorithm (see description below). While this algorithm produces the desired results, it is known to be hyper-exponential: as the length ℓ of the formula increases, its running time grows faster than 2^{2^ℓ}. Thus, from the application viewpoint, it is desirable to come up with a faster algorithm. For some important cases, such faster algorithm was proposed in [9].

4 Proof of the Main Results

Tarski-Seidenberg algorithm: reminder. In this paper, we will use Tarski-Seidenberg algorithm; see, e.g., [1, 8]. This algorithm deals with the *first-order theory of real numbers.* Formulas of this theory are defined as follows:

- we start with real-valued variables $x_1, \ldots, x_n$;
- *elementary formulas* are formulas of the type $F = 0$, $F > 0$, or $F \geq 0$, where F is a polynomial with integer coefficients;
- finally, a general formula can be obtained from elementary formulas by using logical connectives ("and" &, "or" $\vee$, "implies" $\rightarrow$, and "not" $\neg$) and quantifiers over real numbers ($\forall x_i$ and $\exists x_i$).

For example, a formula describing that the given polynomial $F(x_1, \cdots, x_n)$ with integer coefficients has a solution with $x_1 > 0$ is a first-order formula:

$$\exists x_1 \ldots \exists x_n \, ((F(x_1, \cdots, x_n) = 0) \,\&\, (x_1 > 0)).$$

Another example is a formula that show that every quadratic polynomial with non-negative determinant has a solution:

$$\forall a \, \forall b \, \forall c \, ((b^2 - 4a \cdot c \geq 0) \to \exists x \, (a \cdot x^2 + b \cdot x + c = 0)).$$

Tarski designed an algorithm that, given a formula from this theory, returns 0 or 1 depending on whether this formula is true or not.

Seidenberg noticed that Tarski's algorithm works by "eliminating" quantifiers one by one, i.e., by sequentially reducing a given formula to a one with one fewer quantifier. Because of this fact, he showed that we can use a similar construction to reduce each first-order formula with free variables to a quantifier-free form.

Tarski-Seidenberg algorithm: corollary. From the above reduction, it follows that if a formula with free variables has a solution, then it also has an *algebraic* solution, i.e., a solution in which each number is a root of a non-zero polynomial with integer coefficients. Namely, we can reduce the original formula to a quantifier-free formula $F(x_1, \cdots, x_n)$.

The formula $\exists x_2 \ldots \exists x_n \, F(x_1, x_2, \cdots, x_n)$ can be similarly reduced to a quantifier-free expression, i.e., to a combination of equalities and inequalities of the type $P(x_1) = 0$, $P(x_1) > 0$, and $P(x_1) \geq 0$. If one of them is an equality, then we get an algebraic number x_1; if all of them are strict inequalities, then the whole range of values satisfies these inequalities and thus, we can select a rational (hence, algebraic) value from this interval.

Once we plug in the algebraic value x_1 into the original formula, we can then similarly find an algebraic value x_2, etc. – and after n stages, we will get a tuple of algebraic numbers $x_1, \ldots, x_n$ that satisfies the original formula $F(x_1, \cdots, x_n)$.

Proof of Propositions 1 and 2. Let us show that by using the Tarski-Seidenberg algorithm, we can come up with the desired simplification algorithms for proving Propositions 1 and 2.

Let us first prove that the relation between the inputs of a straightforward algorithm and the result(s) of applying this algorithm can be described by a first order formula. Indeed, we have finite number of branchings, so we have finitely many possible branches. On each branch, each value r_i is computed by using elementary arithmetic operations and is, thus, a rational function of the inputs – i.e., ratios of two polynomials. Each branching inequality is thus an inequality between rational functions.

Equalities and inequalities between rational functions can be explained in terms of the polynomials:

- the relation $F/G = 0$ for polynomials F and G is equivalent to $F = 0$, and
- the relation $F/G \geq 0$ for polynomials F and G is equivalent to $((F \geq 0) \,\&\, (G > 0)) \vee ((F \leq 0) \,\&\, (G < 0))$.

Thus, the relation between the inputs and the results of a straightforward algorithm can be indeed described by a first order formula.

In these terms, the first task is to check whether the following formula is true:

$$\exists c_{M+1} \ldots \exists c_N \, \forall y_1 \ldots \forall y_m \, (x = A(y, c) \to P(x, y)),$$

where:

- the first order formula $x = A(y, c)$ means that the tuple $x = (x_1, \cdots, x_n)$ is the result of applying the algorithm A to the values y supplemented by the parameters $c_{M+1}, \ldots, c_N$, and
- the first order formula $P(x, y)$ means that all the relations forming P are satisfied, i.e., that the results of all the straightforward algorithms computing the functions f_i are indeed either equal to 0 or greater than or equal to 0.

We can see that this is also a first order formula. Thus, by using the Tarski algorithm, we can decide whether this formula is true or not.

If this formula is true, then the above Corollary enables us to actually find the values $c_{M+1}, \ldots, c_N$ for which the corresponding formula is true – i.e., for which the algorithm A solves the original data processing problem. This proves Proposition 1.

To prove Proposition 2, it is sufficient to show that the relation between y and each of the bounds $\underline{x}_i$ and $\overline{x}_i$ can also be described by first order formulas. Indeed, the property that $\underline{x}_i$ is the exact lower bound of all the values x_i for which the tuple x is consistent with the observations y can be described in the first order form:

$$\neg\exists x_1 \ldots \exists x_n (x_i < \underline{x}_i \,\&\, P(x,y)) \,\&\, \forall \varepsilon > 0 \,\exists x_1 \ldots \exists x_n\, (x_i < \underline{x}_i + \varepsilon \,\&\, P(x,y)).$$

Similarly, the property that $\overline{x}_i$ is the exact upper bound of all the values x_i for which the tuple x is consistent with the observations y can be described in the first order form:

$$\neg\exists x_1 \ldots \exists x_n (x_i > \overline{x}_i \,\&\, P(x,y)) \,\&\, \forall \varepsilon > 0 \,\exists x_1 \ldots \exists x_n\, (x_i > \overline{x}_i - \varepsilon \,\&\, P(x,y)).$$

Thus, the use of the Tarski-Seidenberg algorithm enables us to prove Proposition 2 as well.

5 Auxiliary Result: In Some Reasonable Sense, the Above Result is the Best We Can Have

Can we do better? The above results use the Tarski-Seidenberg algorithm for deciding first order formulas. In this proof, we spend quite some efforts reducing our problems to the first order formulas. So, a natural question is: maybe there is a larger class of formulas for which a deciding algorithm is possible, a class which would be more natural for our problem.

Towards possible better results. When we think about speeding up algorithms, a natural idea is to take into account that out of four hardware supported elementary arithmetic operations, division is the slowest. Thus, one possibility to speed up computations is to come up with algorithms that do not use division, i.e., with algorithms that only use polynomials and not general rational functions.

Comment. This is, by the way, one of the reasons why, as we have mentioned, $\sin(x)$ and $\exp(x)$ are computed as polynomials (namely, as the sums of the first few terms in the corresponding Taylor expansion).

A seemingly natural idea. In view of the above, a seemingly natural idea is to add the existence of a polynomial to the list of elementary operations. Thus, we arrive at the following definition.

Definition 7. *We will define a* p-language *(p for polynomial) with variables of two types:*

- *variables $x_1, x_2, \ldots$ that run over real numbers, and*
- *for every integer $k > 0$, variables P_1^k, P_2^k, … that run over polynomials of k variables.*

Then:

- p-terms *are defined, by induction, as follows:*
  - *a variable is a p-term;*
  - *the sum, the difference, and the product of p-terms is a p-term;*
  - *if $t_1, \ldots, t_k$ are p-terms, then $P_i^k(t_1, \cdots, t_k)$ is a p-term.*
- *For each p-term t, we can have three types of* elementary p-formulas*: $t = 0$, $t > 0$, and $t \geq 0$.*
- *A* p-preformula *F is any formula can be obtained from elementary formulas by using:*
  - *logical connectives "and" &, "or" $\vee$, "implies" $\rightarrow$, and "not" $\neg$, and*
  - *quantifiers over real numbers: $\forall x_i$ and $\exists x_i$.*
- *Finally, a* p-formula *is a formula of the type $\exists P_i^k \ldots F$, where P_i^k are all polynomial variables in F.*

Example. In these terms, the possibility of polynomial data processing can be described in a very natural way, as

$$\exists P_1^m \ldots \exists P_n^m \,\forall y\, ((x_1 = P_1^m(y) \,\&\, \cdots \,\&\, x_n = P_n^m(y)) \rightarrow P(x,y)).$$

Alas. Unfortunately, for this natural generalization, no deciding algorithm is possible:

Proposition 3. *No algorithm is possible that, given a closed p-formula, checks whether this formula is true or not.*

Proof. Our proof is based on the known result [5] that no algorithm is known that, given an equation $F(v_1, \cdots, v_n) = 0$, where $F(v_1, \cdots, v_n)$ is a polynomial with integer coefficients, checks whether this equation has a solution in which all v_i are natural numbers. This result is known since it solves 10th Hilbert's problem.

Let us show that if it was possible to decide whether a given p-formula holds, then we would be able to checking whether polynomial equations has natural-number solutions – and since this is not possible, this means that it is not possible to decide truth on p-formulas either.

Some coefficients of a polynomial F as positive, some are negative. By moving all the terms with negative coefficients to the other side of the equation $F = 0$, we get an equivalent equation $G(v_1, \cdots, v_n) = H(v_1, \cdots, v_n)$ in which all the coefficients of both polynomials $G(v_1, \cdots, v_n)$ and $H(v_1, \cdots, v_n)$ are positive. Let us form the corresponding p-formula as follows:

1°. To every variable v_i, we put into correspondence a polynomial $P_i^1(x_1)$ for which

$$\forall x_2\, \forall x_3\, (P_i^1(x_2 \cdot x_3) = P_i^1(x_2) \cdot P_i^1(x_3)).$$

One can easily check that all such polynomials have the form $P_i^1(x_1) = x_1^{v_i}$ for some natural number v_i.

2°. To each term v_i^2, we put into correspondence a p-term $P_i^1(P_i^1(x_1))$; one can see that this polynomial has the form $(x_1^{v_i})^{v_i} = x_1^{v_i^2}$.

3°. Similarly, to each term v_i^3, we put into correspondence a p-term $P_i^1(P_i^1(P_i^1(x_1)))$; this term corresponds to $x_1^{v_1^3}$.

4°. In general, to each term v_i^k, we put into correspondence a p-term $P_i^1(\cdots(P_i^1(x_1))\cdots)$ (k times); this term corresponds to $x_1^{v_1^k}$.

5°. To each monomial $m \stackrel{\text{def}}{=} v_i^{k_i} \cdot \ldots v_j^{k_j}$, we similarly put into a correspondence a composition of p-terms corresponding to $v_i^{k_i}$, …, $v_j^{k_j}$. One can easily check that this term has the form x_1^m.

6°. Now, a general polynomial F with positive integer coefficients can be represented as a sum of monomials. For example, the expression $v_1^2 + 2v_1 \cdot v_2 + v_2^2$ can be represented as $v_1^2 + v_1 \cdot v_2 + v_1 \cdot v_2 + v_2^2$.

To each such polynomial, we put into correspondence a p-term F' which is defined as a product of the p-terms corresponding to the monomials. Since $x_1^m \cdot x_1^{m'} = x_1^{m+m'}$, the value of this term is exactly x_1^F.

7°. Now, we can form a p-formula

$$\exists P_1^1 \ldots \exists P_n^1\, (\forall x_2\, \forall x_3\, (P_1^1(x_2 \cdot x_3) = P_1^1(x_2) \cdot P_1^1(x_3)\, \&\, \cdots\, \&\, P_n^1(x_2 \cdot x_3) = P_n^1(x_2) \cdot P_n^1(x_3))\, \&\, G'(2) = H'(2)),$$

where G' and H' are p-terms corresponding to the polynomials G and H. The existence of such polynomials means the existence of natural numbers v_i for which $P_i^1(x_1) = x_1^{v_i}$, and the equality $G'(2) = H'(2)$ means that $2^G = 2^H$, i.e., that $G = H$. Thus, the truth of this formula implies that the original equation $F = 0$ has a solution in natural numbers.

Vice versa, if this equal has natural-valued solution $v_1, \ldots, v_n$, then the polynomials $P_i^1(x_1) = x_1^{v_i}$ satisfy the above formula.

The reduction is proven, and so is the proposition.

Acknowledgments

This work was supported in part by the National Science Foundation grants HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721.

References

[1] Basu, S., Pollack, R., and M.-F. Roy, *Algorithms in Real Algebraic Geometry*, Springer-Verlag, Berlin, 2006.

[2] Jaulin, L., Kieffer, M., Didrit, O., and E. Walter, *Applied Interval Analysis*, Springer Verlag, London, 2001.

[3] Klir, G., and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

[4] Kreinovich, V., Lakeyev, A., Rohn, J., and P. Kahl, *Computational Complexity and Feasibility of Data processing and Interval Computations*, Kluwer, Dordrecht, 1997.

[5] Matiyasevich, Y.V., *Hilbert's Tenth Problem*, MIT Press, Cambridge, Massachusetts, 1993.

[6] Moore, R.E., Kearfott, R.B., and M.J. Cloud, *Introduction to Interval Analysis*, SIAM Press, Philadelphia, Pennsylviania, 2009.

[7] Nguyen, H.T., and E.A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2006.

[8] Tarski, A., *A Decision Method for Elementary Algebra and Geometry*, 2nd Edition, Berkeley and Los Angeles, 1951.

[9] Urenda, J., *Algorithmic Aspects of the Embedding Problem*, Ph.D. Dissertation, Department of Mathematical Sciences, New Mexico State University, Las Cruces, New Mexico, 2015.

[10] Zadeh, L.A., Fuzzy sets, *Information and Control*, vol.8, pp.338–353, 1965.